

LSPS Expression Language 3.3

1 Types and values

1.1 Simple

String	"string" #ASCII_CODE #"not-localized string" \$localizedString(argument1, argument2,...)
Boolean	true false
Integer	1 -100
Decimal	24.86 6.63E-34 Decimal(scale) Decimal(scale, rounding)
Date	d'yyyy-MM-dd HH:mm:ss.SSS' date(year, month, day) date(year, month, day, hours, minutes, seconds, millis) date(string) date(string, pattern)
LocalDate	ld'yyyy-MM-dd' localDate(epochDay) localDate(string) date(year, month, day)
Object	any expression
Null	null

1.2 Containers

Set<T>	{ element1,... }
List<T>	[element1,...]
List<Integer>	-5..81 1000..0
Map<K,V>	[key1 -> value1,...] empty map: [->]

1.3 Other

Record	new record_name() new record_name(value1, value2,...) new record_name(field2 -> value2, field4 -> value4,...)
Reference<T>	&variable
Closure {T...: U}	{ -> expression } { x:Type, y -> expression }
Type	type(type_definition) record_name

2 Operators

2.1 Assignment

Assignment	variable := expression
------------	------------------------

2.2 Arithmetic

Addition	+
Subtraction	-
Increment	++var var++
Decrement	--var var--
Multiplication	*
Division	/
Modulo	%
Exponentiation	**

2.3 Boolean

Equality	== != <>
Comparison	< <= > >= <=> instanceof
Negation	not !
Conjunction	and &&
Disjunction	or
Exclusive disjunction	xor
Pattern matching	string like pattern
Collection containment	item in collection

2.4 Other

String concatenation	+
Parenthetical grouping (precedence)	()

2.5 Compound Assignment

Addition	+=
Subtraction	-=
Multiplication	*=
Division	/=
Modulo	%=

3 Access

Set element	set[index]
List element	list[index]
Map element	map[key]
Record field	record.field
Record safe field	record?.field
Record reference field	reference.field
Record ref. safe field	reference?.field
Enumeration literal	enum.literal
Dereferencing	*reference
Namespace separator	::
Function call	function T1,... (argument1, argument2,...) function T1,... (name1->value1, name2->value2,...)
Closure invocation	closure(argument1, argument2,...)
Method call	object.method(argument1, argument2,...)

4 Special constructs

Local variable	<code>def type name</code> with assignment: <code>def type name := expression</code>
Comments	<code>// single-line comment</code> <code>/*</code> multi-line comment <code>*/</code>
Assignment	<code>variable := expression</code>
Expression variable	<code>def type name</code> final variable: <code>final type name</code> with assignment: <code>def type name := expression</code>
Expression chaining	<code>expr1 ; expr2 ; expr3 ; ...</code>
Block	<code>begin expression end</code>
Error throw	<code>error(string_expression)</code>
Try-catch	<code>try expression</code> <code>catch exception,... -> expression</code> <code>...</code> <code>end</code>
Cast	<code>expression as type</code>

5 Control flow

If-then	<code>if condition1 then</code> <code>expression1</code>
If-then-else	<code>expression1</code>
If-then-elsif-then	<code>elsif condition2 then</code> <code>expression2</code>
If-then-elsif-then-e	<code>...</code> <code>else</code> <code>expressionN</code> <code>end</code>
Ternary conditional	<code>condition ? expression1 :</code> <code>expression2</code>
Null-coalescing	<code>expression1 ?? expression2</code>
Switch	<code>switch arg_expression</code> <code>case c1, c2,... -> expression1</code> <code>...</code> <code>default -> defaultExpression</code> <code>end</code>

While looping	<code>while condition do</code> <code>expression</code> <code>end</code>
For looping	<code>for(init; condition; update) do</code> <code>expression</code> <code>end</code>
Collection iteration	<code>foreach type iterator in collection</code> <code>do</code> <code>expression</code> <code>end</code>
Looping break (in while,for and foreach)	<code>break</code>
Looping iteration skipping (in while, for and foreach)	<code>continue</code>

6 Function definition file

Function	<code>/** description */</code> <code>annotations</code> <code>visibility <T,...> ReturnType</code> <code>function(Type1 param1,...)</code> <code>body</code>
Function body	<code>expression: { expression }</code> <code>native: native java_method_path</code>
Annotations	<code>@SideEffect</code> <code>@Deprecated</code> <code>@Disabled</code> <code>@Status(statusName)</code> <code>@Meta(key1->value1,</code> <code>key2->value2,...)</code>
Visibility	<code>public</code> <code>private</code>
Function parameter	optional: Type name mandatory: Type name* variadic: Type... name default value: Type name = defaultValueExpression

7 Method definition file

Record methods	<code>Record {</code> <code>methods</code> <code>}</code>
Method	<code>/** description */</code> <code>annotations</code> <code>visibility modifiers <T,...></code> <code>ReturnType function(Type1</code> <code>param1, Type2 param2*,...)</code> <code>body</code> Note: Annotations and parameters follow syntax of function.
Constructor	<code>/** description */</code> <code>annotations</code> <code>visibility Record(Type1 param1,</code> <code>Type2 param2*,...)</code> <code>body</code> Note: Annotations, parameters and body follow syntax of function; visibility follows syntax of method.
Method body	<code>expression: { expression }</code> <code>native: native java_method_path</code> <code>abstract: ;</code>
Visibility	<code>public</code> <code>private</code> <code>protected</code>
Modifiers	<code>abstract</code> <code>static</code>

7.1 Access (only in method)

This instance	<code>this</code>
Super-type members	<code>super.method(argument,...)</code>

7.2 Access (only as first statement of constructor)

Calling constructor of this record	<code>this(argument,...)</code>
Calling constructor of supertype	<code>super(argument,...)</code>